# Problem 1. Data Modeling

### [Q. 1]

**1-1.** Fill in the blanks in the provided tables.

**1-2.** Assign PK to all tables.

- *(Answer)*
    - *Table with filled-in blanks*
    - *Table with PK column*

### [Q. 2]

Design a data pipeline (such as an Airflow DAG) based on the derived tables to generate KPI data, and develop the pipeline code at a pseudocode level. Provide a detailed explanation of the logic behind each task, the relationships between tables, and identify potential considerations and precautions that may arise during data processing.

- *(Answer)*
    - *Data pipeline flow chart*
    - *Pipeline pseudocode*
    - *Document outlining the logic considered during the pipeline configuration and highlighting key points to be noted.*
    - *SQL used to create the pipeline*

### [Q. 3]

Specify the index (table of contents) of the data document that external teams can use for various analytical purposes, and provide an explanation of the reasons behind such an index definition.

- *(Answer)*
    - *Index of the document, what content is included, and a document that explains why you think so*

## [Service Features]

Company H has launched a dating service app. The primary function of this app is swiping the screen to initiate video chat  with people from around the world.

The service can be used for free, and also users can purchase a "Premium Select" item to match with people of a specific country or gender.

The subscription service has a validity period of one month, provides 50 “Premium Select” items, and also provides 10 Premium Selects per day. During the subscription period, Premium Selects can be purchased at a 20% discounted price.

Items can be purchased with local currency through various markets (Play Store, App Store) or third-party payment methods (e.g., Toss). The price of the "Premium Select" is 1000 KRW for a purchase made in Korea, and item prices may vary depending on the country or duration.

Users can get "Premium Selects" for free through campaigns, but they will expire if not used within 10 days of receipt.
However, only one campaign can be participated in, and only one item can be obtained.

## [KPI Layout]

1. Revenue in USD and KRW and the number of purchasers by country and market (Android, iOS)
2. Revenue trend in USD and KRW for subscription and non-subscription products
3. Refund trend in USD and KRW for each item

## [KPI Requirements]

1. Users can sign in to various countries during the day. (Proper criteria need to be defined in the KPI)
2. Define the country processing criteria and explain the reason for the definition.
3. Exchange rates should be reflected at the exchange rate of the respective date.

## [Pipeline Design Conditions]

1. The service DB table data is collected to a data warehouse in the same format table at one hour intervals.
2. Exchange rates are collected once a day at 09:00 KST with the previous day's data.
3. Define aggregation tables for KPI data processing.
4. KPI data is aggregated on a daily basis.
5. Please design a data pipeline considering cost optimization.

## [Table schema]

### user

| | |
|---|---|
| user_id | integer |
| country | string |
| gender | string |
| last_login_datetime | datetime |
| create_datetime | datetime |
| update_datetime | datetime |

### order

| | |
|---|---|
| order_id | string |
| order_datetime | datetime |
| campaign_id | string |
| order_type(item / subs) | string |
| user_id | integer |
| order_status(sales / refund) | string |
| purchase_market_cd | string |
| payment_method_cd | string |
| currency | string |
| amount | float |
| create_datetime | datetime |
| update_datetime | datetime |

### order_detail

| | |
|---|---|
| order_id | string |
| unit_purchase_amount | float |
| item_qty | integer |
| create_datetime | datetime |
| update_datetime | datetime |

### order_mapping

| | |
|---|---|
| order_id | string |
| map_order_id | string |
| discount_rate | float |
| create_datetime | datetime |
| update_datetime | datetime |

### item

| | |
|---|---|
| item_id | string |
| item_name | string |
| expiration_date | integer |
| create_datetime | datetime |
| update_datetime | datetime |

### item_price

| | |
|---|---|
| item_id | string |
| country | string |
| price | float |
| create_datetime | datetime |
| update_datetime | datetime |

### campaign

| | |
|---|---|
| campaign_id | string |
| campaign_name | string |
| item_id | string |
| create_datetime | datetime |
| update_datetime | datetime |

### exchange

### sales_kpi

# Problem 2. Airflow DAG

## [Q. 1]

Create an Airflow DAG according to the conditions below.

## [A table schema]

dt: date
hr: int
value: string(json)

There is a table A with the above schema. The value is a json string.
key: id, value: int
key: user_name, value: string
ex) {“id”: 1, “user_name”: “test_user”}

Using a sensor, check if there is the previous day's data in table A, and if there is data, parse the json value to create a table B with the schema of dt, hr, id, and user_name.

## [Conditions]

1. The DAG is scheduled at 10 AM KST.
2. You can use any environment you are familiar with for setting up and testing the table (BigQuery, MySQL, PostgreSQL, etc.)
    a. You can use cloud, set up your own DB, or use Docker, etc...

# Problem 3. Data Ingestion Type

## [Q. 1]

Data backfilling involves changing or modifying historical data. The source data in this case has already undergone processing methods such as full_dump, segmented, and upsert, as described in the [Background]. Some issues have been discovered within this processed source data, which now requires a data backfill operation. Your task is to outline the backfilling process for each of the full_dump, segmented, and upsert types, while considering the following points:

1. The range of the backfill operation involves a portion of data that occurred between 2023-04-01 and 2023-04-30.
2. The range of modifications made during the backfill process should be as small as possible to optimize the cost.
3. Provide clear explanations and example SQLs for each type of data ingestion method involved in the backfill process.

Please refer to the [Reference Table Schema] section for the schema of each table involved in this problem.

## [Background]

There are three types of tables for data ingestion based on their action:

### 1. full_dump

a. Reflects the entire source table.
b. This type of table operates on the entire data set without specifying a range. When the query is executed, it scans the entire table and returns all data.

### 2. segmented

a. This type incrementally ingests data by specific date or range based on a partition column (e.g., partition_date).
b. By using the partition column, it processes only a portion of the data, reducing the workload on the entire data set. It is used for adding or updating data incrementally.

### 3. upsert

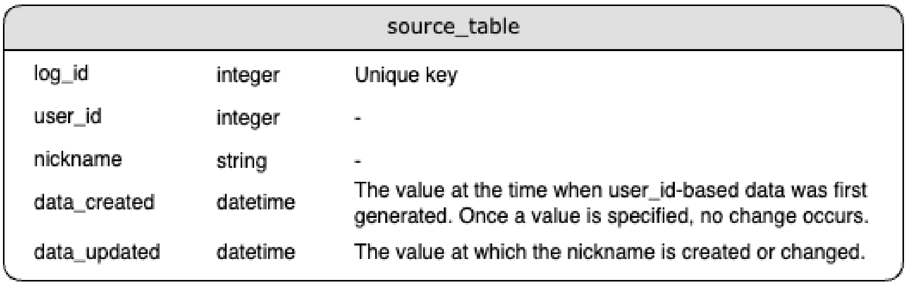
a. This type modifies (or adds) only the necessary data.
b. It inserts or updates data based on a specific key column. If an existing key value is present, the data is updated; otherwise, new data is inserted.

## [Reference Table Schema]

### 1. source_table

a. This is a log-type table that records changes in nickname based on the user_id.

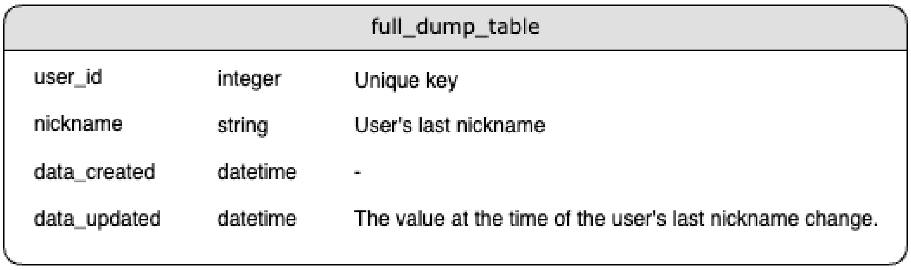
b. All columns are not null.
c. data_created is the timestamp when the data is first generated based on user_id.
d. data_updated is the timestamp when the nickname is changed.



| source_table | | |
|---|---|---|
| log_id | integer | Unique key |
| user_id | integer | - |
| nickname | string | - |
| data_created | datetime | The value at the time when user_id-based data was first generated. Once a value is specified, no change occurs. |
| data_updated | datetime | The value at which the nickname is created or changed. |

## 2. full_dump_table (full_dump type)

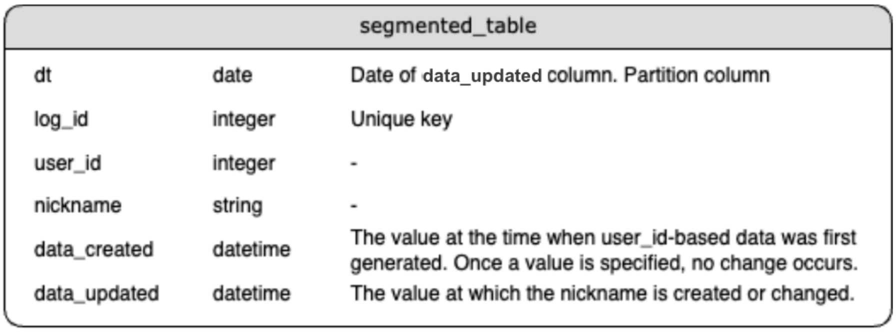
a. This table ingests data from the source table.
b. Based on user_id, it aggregates the latest nickname data and stores it as a full_dump type table.



| full_dump_table | | |
|---|---|---|
| user_id | integer | Unique key |
| nickname | string | User's last nickname |
| data_created | datetime | - |
| data_updated | datetime | The value at the time of the user's last nickname change. |

## 3. segmented_table (segmented type)

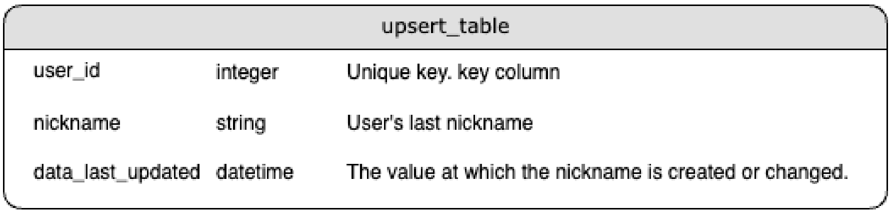
a. This table ingests data from the source table.
b. It is stored as a segmented type table.
c. The dt column is the Partition column.



| segmented_table | | |
|---|---|---|
| dt | date | Date of data_updated column. Partition column |
| log_id | integer | Unique key |
| user_id | integer | - |
| nickname | string | - |
| data_created | datetime | The value at the time when user_id-based data was first generated. Once a value is specified, no change occurs. |
| data_updated | datetime | The value at which the nickname is created or changed. |

## 4. upsert_table (upsert type)

a. This table ingests data from the source table.
b. The user_id column is the key column.
c. It is an upsert type table that aggregates the latest nickname data based on user_id.



| upsert_table | | |
|---|---|---|
| user_id | integer | Unique key. key column |
| nickname | string | User's last nickname |
| data_last_updated | datetime | The value at which the nickname is created or changed. |

# Problem 4. History master loading

## [Q. 1]

Write a query that generates a device_history table using the user_device_log table, taking into consideration the [Situation] and [Conditions] provided below.

## [Situation]

You want to create a device_history table based on users' device information when they log in. The table should be derived from the user_device_log table, and organized according to the user_id.

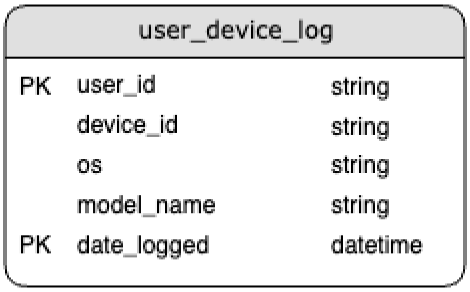
The device_history table should store the change history of data with fixed columns (user_id) and columns where changes occur (device_id, os, model_name). These changes should be specified using start_datetime and end_datetime.

- user_device_log
  - Primary Key (PK) - user_id, date_logged
  - All values do not have nulls.
  - log data.
  - Data is usually obtained in chronological order, but does not guarantee the order of data acquisition.
- device_history
  - Primary Key (PK) - user_id, start_datetime
  - All values do not have nulls.

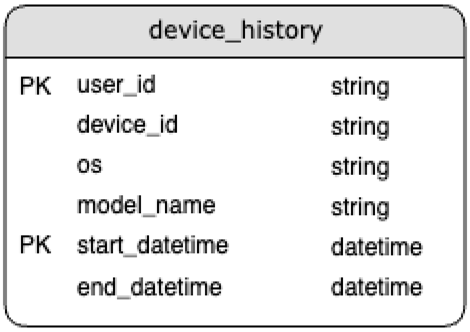
The update logic in the device_history table should follow the Slowly Changing Dimension (SCD) Type 2 method.

## [Reference]

### [tables schema]



| user_device_log | | |
|---|---|---|
| PK | user_id | string |
| | device_id | string |
| | os | string |
| | model_name | string |
| PK | date_logged | datetime |



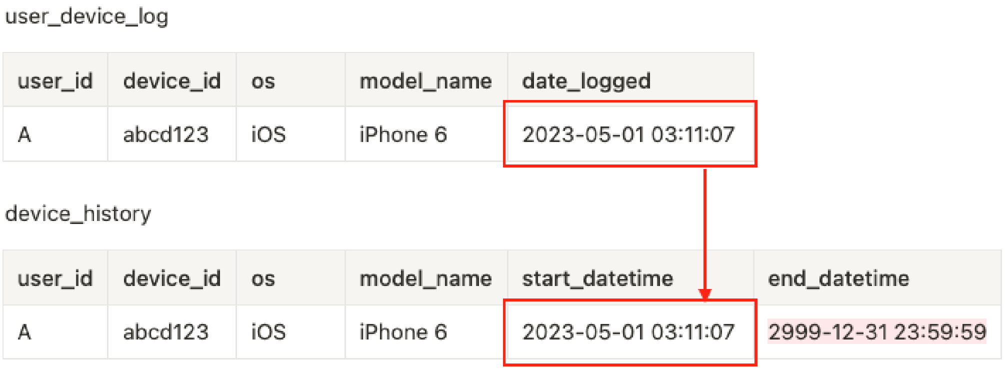
| device_history | | |
|---|---|---|
| PK | user_id | string |
| | device_id | string |
| | os | string |
| | model_name | string |
| PK | start_datetime | datetime |
| | end_datetime | datetime |

Sql For Test

### [Update logic for the device_history table using SCD Type 2]

user_device_log
device_history

Consider the following detailed example to illustrate the update logic of the device_history table with the user_device_log data:



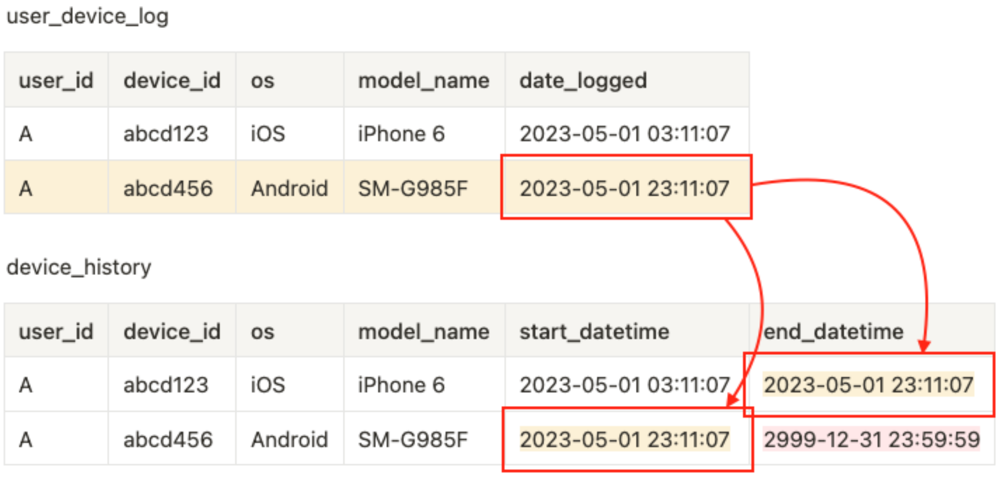
user_device_log

| user_id | device_id | os | model_name | date_logged |
|---|---|---|---|---|
| A | abcd123 | iOS | iPhone 6 | 2023-05-01 03:11:07 |

device_history

| user_id | device_id | os | model_name | start_datetime | end_datetime |
|---|---|---|---|---|---|
| A | abcd123 | iOS | iPhone 6 | 2023-05-01 03:11:07 | 2999-12-31 23:59:59 |

1. When user_id = 'A' first appears, the device_history table should be loaded as illustrated in the example below:
    a. The start_datetime of the new row should be set to the log acquisition time (date_logged), and the end_datetime should be fixed to the distant future, specifically '2999-12-31 23:59:59'.



user_device_log

| user_id | device_id | os | model_name | date_logged |
|---|---|---|---|---|
| A | abcd123 | iOS | iPhone 6 | 2023-05-01 03:11:07 |
| A | abcd456 | Android | SM-G985F | 2023-05-01 23:11:07 |

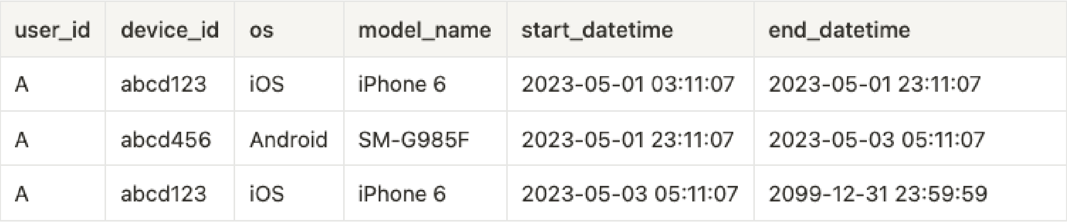
device_history

| user_id | device_id | os | model_name | start_datetime | end_datetime |
|---|---|---|---|---|---|
| A | abcd123 | iOS | iPhone 6 | 2023-05-01 03:11:07 | 2023-05-01 23:11:07 |
| A | abcd456 | Android | SM-G985F | 2023-05-01 23:11:07 | 2999-12-31 23:59:59 |

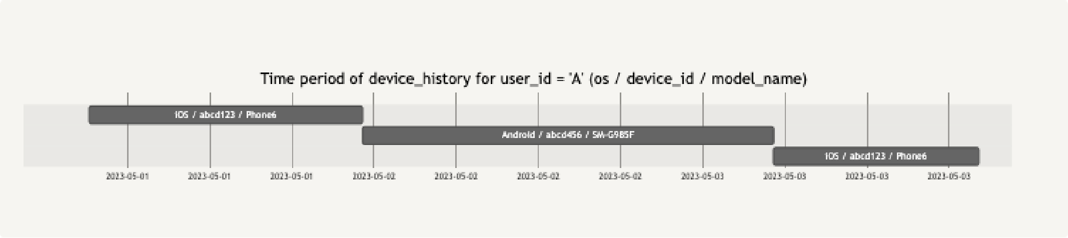
2. Subsequently, when a user_device_log containing new device information for user_id = 'A' appears, the device_history table should be updated as follows:
    a. Compare the previously loaded value in user_device_log with the value in the changing column (device_id, os, model_name) found in the new log.
    b. If the value changes, create a new row in the user_device_log table.
    c. The start_datetime of the new row should be the date_logged when the modified log was obtained, and the end_datetime should be fixed to the distant future, specifically '2999-12-31 23:59:59'.
    d. Additionally, update the end_datetime in the old row of the new row to match the start_datetime of the new row (i.e., the start_datetime of the new row is the same as the end_datetime of the previous row).

## [Conditions]

- Based on a fixed column (ex. user_id), there should be no missing interval between the end_datetime of the current row and the start_datetime of the next row.
- When you list start_datetime, end_datetime based on a fixed column (ex. user_id), there should be no overlapping time.
- Please don’t use trigger



| user_id | device_id | os | model_name | start_datetime | end_datetime |
|---|---|---|---|---|---|
| A | abcd123 | iOS | iPhone 6 | 2023-05-01 03:11:07 | 2023-05-01 23:11:07 |
| A | abcd456 | Android | SM-G985F | 2023-05-01 23:11:07 | 2023-05-03 05:11:07 |
| A | abcd123 | iOS | iPhone 6 | 2023-05-03 05:11:07 | 2099-12-31 23:59:59 |



# Sql For Test

Unset

```
CREATE TABLE user_device_log
(
    user_id     varchar(255)   NOT NULL,
    device_id   varchar(255)   NOT NULL,
    os          varchar(255)   NOT NULL,
    model_name  varchar(255)   NOT NULL,
    date_logged datetime       NOT NULL
)
;

CREATE TABLE device_history
(
    user_id        varchar(255)   NOT NULL,
    device_id      varchar(255)   NOT NULL,
    os             varchar(255)   NOT NULL,
    model_name     varchar(255)   NOT NULL,
    start_datetime datetime       NOT NULL,
    end_datetime   datetime       NOT NULL
)
;

INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A','abcd123','iOS','iPhone 6','2023-05-01 00:11:07');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd123', 'iOS', 'iPhone 6', '2023-05-01 03:11:07');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd123', 'iOS', 'iPhone 6', '2023-05-01 05:05:14');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd123', 'iOS', 'iPhone 6', '2023-05-01 11:17:28');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd456', 'Android', 'SM-G985F', '2023-05-01 23:11:07');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg123', 'Android', 'SM-A333F', '2023-05-02 00:14:17');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd456', 'Android', 'SM-G985F', '2023-05-02 04:14:33');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'hijk765', 'iOS', 'iPhone 7 Plus', '2023-05-02 22:12:51');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd456', 'Android', 'SM-G985F', '2023-05-03 05:07:38');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd123', 'iOS', 'iPhone 6', '2023-05-03 05:11:07');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'hijk765', 'iOS', 'iPhone 7 Plus', '2023-05-03 05:13:24');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd123', 'iOS', 'iPhone 6', '2023-05-04 04:14:34');
```

```
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg123', 'Android', 'SM-A333F', '2023-05-04 11:15:50');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'hijk765', 'iOS', 'iPhone 7 Plus', '2023-05-04 15:13:57');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd123', 'iOS', 'iPhone 6', '2023-05-05 05:07:39');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd123', 'iOS', 'iPhone 6', '2023-05-05 05:11:08');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg123', 'Android', 'SM-A107F', '2023-05-05 05:13:17');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'hijk765', 'iOS', 'iPhone 7 Plus', '2023-05-05 05:13:25');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg123', 'Android', 'SM-A107F', '2023-05-06 15:13:50');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd456', 'Android', 'SM-G985F', '2023-05-07 04:14:35');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd456', 'Android', 'SM-G985F', '2023-05-07 05:07:40');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'hijk765', 'iOS', 'iPhone 7 Plus', '2023-05-07 22:12:52');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg123', 'Android', 'SM-A333F', '2023-05-08 04:16:18');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg123', 'Android', 'SM-A107F', '2023-05-08 05:13:18');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'hijk987', 'iOS', 'iPhone X', '2023-05-09 19:11:57');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg123', 'Android', 'SM-A107F', '2023-05-10 15:13:51');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'hijk987', 'iOS', 'iPhone X', '2023-05-10 22:12:53');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg123', 'Android', 'SM-A333F', '2023-05-11 15:17:51');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'hijk987', 'iOS', 'iPhone X', '2023-05-11 19:11:58');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'hijk987', 'iOS', 'iPhone X', '2023-05-11 22:12:54');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg124', 'Android', 'SM-G610F', '2023-05-12 04:03:19');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'hijk987', 'iOS', 'iPhone X', '2023-05-12 19:11:59');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg124', 'Android', 'SM-G610F', '2023-05-13 11:23:12');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg124', 'Android', 'SM-G610F', '2023-05-14 05:13:19');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg124', 'Android', 'SM-G610F', '2023-05-15 15:13:52');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg123', 'Android', 'SM-A107F', '2023-05-16 05:13:20');
```

```
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg123', 'Android', 'SM-A333F', '2023-05-17 09:16:20');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg124', 'Android', 'SM-G610F', '2023-05-17 15:13:53');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg124', 'Android', 'SM-G610F', '2023-05-17 17:17:43');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg124', 'Android', 'SM-G610F', '2023-05-20 05:13:21');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg124', 'Android', 'SM-G610F', '2023-05-20 14:34:34');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg124', 'Android', 'SM-G610F', '2023-05-20 15:13:54');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'hijk987', 'iOS', 'iPhone X', '2023-05-20 22:12:55');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg124', 'Android', 'SM-G610F', '2023-05-21 03:23:21');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg177', 'Android', 'SM-J415F', '2023-05-21 05:13:22');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg177', 'Android', 'SM-J415F', '2023-05-21 07:45:22');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'lmn900', 'iOS', 'iPhone 11', '2023-05-21 22:12:55');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'lmn900', 'iOS', 'iPhone 11', '2023-05-22 22:12:53');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg177', 'Android', 'SM-J415F', '2023-05-23 15:13:55');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg177', 'Android', 'SM-J415F', '2023-05-23 15:53:25');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'lmn900', 'iOS', 'iPhone 11', '2023-05-23 19:11:58');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('C', 'lmn900', 'iOS', 'iPhone 11', '2023-05-23 22:12:54');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg177', 'Android', 'SM-J415F', '2023-05-24 05:03:23');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg177', 'Android', 'SM-J415F', '2023-05-24 05:13:23');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd123', 'iOS', 'iPhone 6', '2023-05-25 05:11:09');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg177', 'Android', 'SM-J415F', '2023-05-25 13:17:36');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd123', 'iOS', 'iPhone 6', '2023-05-26 04:14:36');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd789', 'iOS', 'iPhone 7 Plus', '2023-05-26 05:07:41');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd789', 'iOS', 'iPhone 7 Plus', '2023-05-26 05:11:10');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg177', 'Android', 'SM-J415F', '2023-05-26 15:13:56');
```

```
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('A', 'abcd789', 'iOS', 'iPhone 7 Plus', '2023-05-28 04:14:37');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('B', 'efg177', 'Android', 'SM-J415F', '2023-05-30 05:13:24');
INSERT INTO user_device_log(user_id,device_id,os,model_name,date_logged)
VALUES('D', 'efg177', 'Android', 'SM-J415F', '2023-05-30 05:18:24');
```

### [비밀유지서약서]

본인은 유한책임회사 하이퍼커넥트를 통해 취득한 코딩테스트/과제물 및 코딩과제/과제물의 형식 및 내용에 관하여 본인과 유한책임회사 하이퍼커넥트 이외 제3자에게 상기 코딩테스트 및 코딩과제 관련 모든 정보를 어떠한 방법으로도 누설하거나 공개하지 않겠습니다. 본인은 본 서약을 성실히 준수할 것을 확인하며, 서약한 사항을 위반한 경우 일체의 민, 형사상 책임을 부담하겠습니다.

본 과제물을 수행함에 따라 본인은 자유로운 의사에 따라 본 서약서의 내용을 묵시적으로 동의하며 어떠한 이의도 없음을 확인합니다.